**Problem 0.** For both programs, I use Python 2.7.6. You can execute cnf converter by executing cnf/main.py, and nonogram solver by executing nonogram/main.py

**Problem 1.** I just simply use the algorithm on the lecture notes. IMPLFREE, NNF, CNF and DISTR functions are used for convert a propositional logic formula to the CNF. To check validity of the input formula, I just negate the formula, change to the CNF, and feed to minisat.

**Problem 2.** My program works in two steps: i) preprocessing, ii) convert to SAT, feed to minisat and get results.
(1) Preprocessing: for some rows and columns, we could know the exact solution for the part of the rows or columns. For example, if there is a row which marked as 15 in a 20 by 20 board, we can easily conclude that the middle 10 cells should be filled. Therefore, i) the program scans all possible configurations for each row and columns, and marks 0 or 1 to some cells that shared same state by all configurations of a row or a column. After scanning all rows and columns, ii) the program removes configurations that do not match with one or more columns or rows configurations. The program repeats the step i) and ii) until no differences, i.e no configurations are removed or no cells are newly marked as 0 or 1.
(2) Convert to SAT: I use only variables $x_{ij}$, which denotes whether a cell of the $i$th row and $j$th column is filled or not. For each rows and columns, the program simply converts all possible configurations that was filtered in step (1) to DNF. Then, it converts the formula to CNF and conjunct them. For example, in the puzzle (i), no information could be obtained in (1), so the program converts every possible configuration of each rows and columns into the formula such as $((x_{11} \wedge \neg x_{12}) \vee (\neg x_{11} \wedge x_{12}))$ for the first row. In the conversion, cells that are marked 0 or 1 in the preprocessing steps are excluded.

**Puzzle 1.** 2;2;1 1;1 1;1 1;1 1; (semicolon denotes the newline character)